BlueGene/L
Single Node Performance Issues

Siddhartha Chatterjee (sc@us.ibm.com)
Kenneth Dockser
John A. Gunnels
Manish Gupta
Fred G. Gustavson
Mark Mendell
James C. Sexton
T. J. C. Ward

IBM (Raleigh, Toronto, Yorktown)

Outline

- Single-core architecture review
  - Dual FPU
  - Memory hierarchy
- Performance issues
  - Memory issues
  - FPU issues
  - Dual-core issues
- Available programming options
- Case study: DAXPY
- Case study: Matrix multiplication

Disclaimer

- All performance projections are preliminary and subject to change
- Performance estimates come from a variety of sources:
  - Simulations on MTI VHDL simulator
  - Simulations on BGLsim simulator
  - Numbers provided by hardware designers
  - Best practice estimates from algorithm designers

Single Core Architecture
Dual FPU

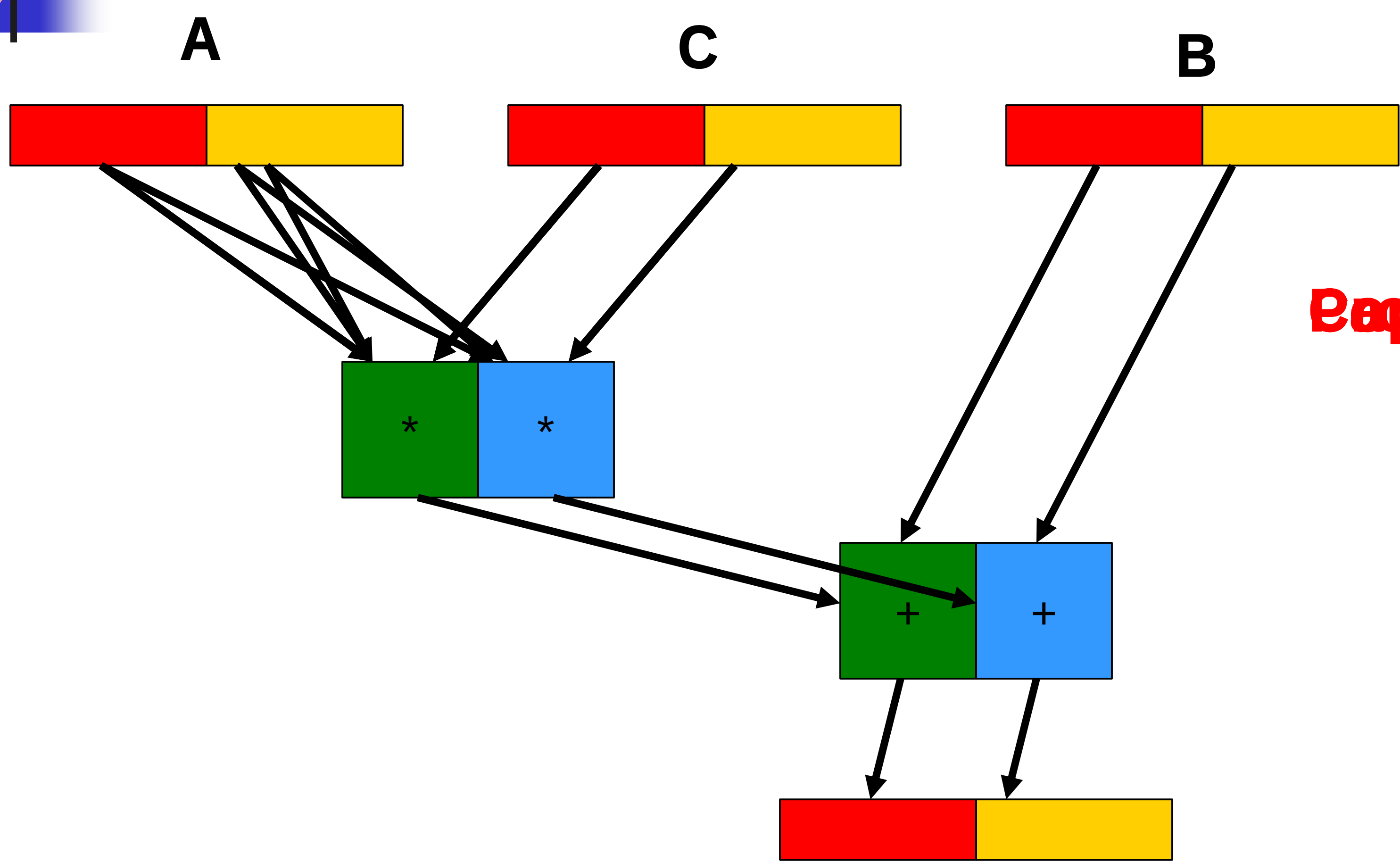
- Two 32-element 64-bit register files
  - Primary (P), secondary (S) registers individually addressable
  - Register pair (P_i, S_i) jointly used in SIMD operations
- Dual floating-point ALU
  - Based on SIMD FMAs
  - Primary FPU used for scalar operations; both FPUs used for SIMD operations
  - All computational operations are double-precision only
  - No support for defining exceptions, exception handlers, and status flags
  - Results conform to IEEE 754 behavior when exceptions are disabled

Single Core Architecture
Dual FPU Instructions

- 2-way SIMD extensions of elementary arithmetic instructions
  - Add, subtract, multiply, reciprocal estimate, reciprocal square root estimate
- 2-way SIMD extensions of FMA ops (T = A*C+B)
  - Parallel
  - Cross
  - Copy-primary
  - Copy-secondary

Single Core Architecture

SIMD FMA Details



Single Core Architecture
More FPU Instructions

- Asymmetric and complex FMAs
  - Primary and secondary FPUs perform related but non-identical operations
  - Useful for performing operations such as FFT butterfly operation and complex arithmetic in general
- Select operations
- Register-register move operations
- Conversion and rounding operations

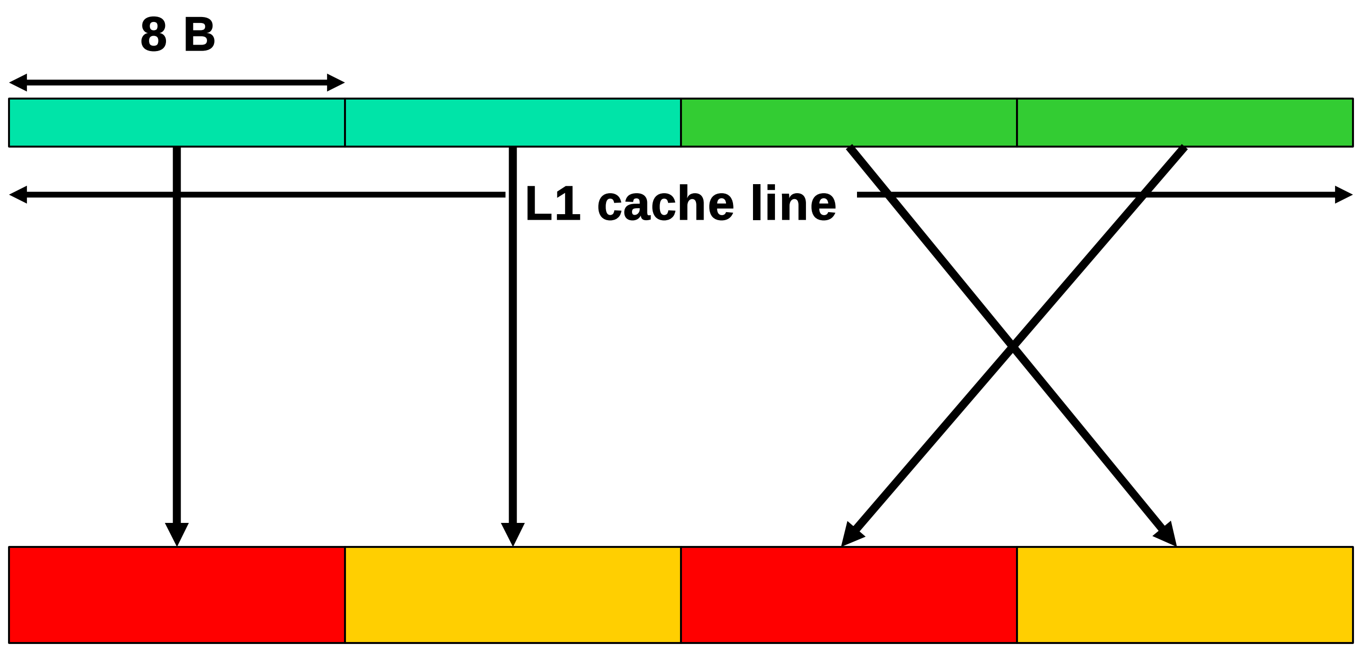
Single Core Architecture

FPU-Memory Interface

- Load/store one double-precision number (doubleword access)
  - To/from primary register
  - To/from secondary register
  - Lower bandwidth, more instructions, greater flexibility
- Load/store two double-precision numbers (quadword access)
  - Parallel
  - Cross
  - Higher bandwidth, fewer instructions, less flexibility

Single Core Architecture

FPU-Memory Interface



- **EA for QW access must be aligned on 128-bit (16 B) boundary**
- **Registers accessed in QW L/S must be a Primary-Secondary pair**

Single Core Architecture
Unit Latencies

- All non-memory operations have def-to-use latency of 5 pclks
- Memory loads have load-to-use latency of 4 pclks (assuming L1 cache hit)
- Memory stores have 3 pclk latency to completion
- Can initiate one memory operation and one FP operation in each cycle
- There is no register renaming in hardware
    - Need to unroll to software pipeline

Programming Options
Low level

- In-line assembly (gnu only)
  - User responsible for instruction selection, register allocation, and scheduling
- Double Hummer intrinsics (XL only)
  - Complex data type used to model pair of double-precision numbers that occupy a (P, S) register pair
  - User responsible for instruction selection
  - Compiler responsible for register allocation and scheduling
  - Supported in C99 and Fortran, not in C++

Programming Options
High Level

- Compiler optimization to find SIMD parallelism (XL only)
  - Currently uses Larsen-Amarasinghe “Superword Level Parallelism” algorithm (PLDI’00) to detect and generate SIMD operations
  - Needs user input for specifying memory alignment and lack of aliasing
    - `__alignx` assertion
    - `disjoint` pragma
  - Currently limited to parallel SIMD and memory operations

Single Node Performance
Memory Issues

- DW vs. QW accesses
  - Misalignment trap is very expensive; program defensively, especially for libraries
- L1 line size is 32 bytes
  - 4 elements / line, 2 QW accesses / line
  - Use single-precision if appropriate (8 elements / line)
- L1 cache issues
  - 32 KB capacity, 64-way associative, round-robin replacement within categories
  - Sets can be split into locked, transient, and normal ways (caution: requires supervisor mode)
- L2, L3, main memory issues
  - Prefetching of streams

Single Node Performance
FPU Issues

- Register organization
  - 64 64-bit registers, organized as 32×2
  - Tricky but possible to use as 64 registers
  - Consciously tile for registers
- Lack of register renaming
  - Increases register usage in SWP'd loops
- Effective use of FP operations
  - Asymmetric and complex FMAs are powerful

Single Node Performance
Dual-Core Issues

- Cores have symmetric access to communication devices
- L1 caches are not coherent between cores
- Possible operation modes
  - Heater mode
  - Communication coprocessor mode
  - Symmetric mode

Programming Example

DAXPY

```
for (i=0; i<n; i++) {
    y[i] = a*x[i]+y[i];
}
```

1:(P0,S0) = LD(x[i],x[i+1])

2:(P1,S1) = LD(y[i],y[i+1])

3:

4:

5:

6: **(P2,S2) = P8*(P0,S0)+(P1,S1)**

7:

8:

9:

10:

11:(y[i],y[i+1]) = ST(P2,S2)

```
for (i=0; i<n; i+=2) {
    y[i] = a*x[i]+y[i];
    y[i+1] = a*x[i+1]+y[i+1];
}
```

Alignment Issues

DAXPY

X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	X[6]	X[7]

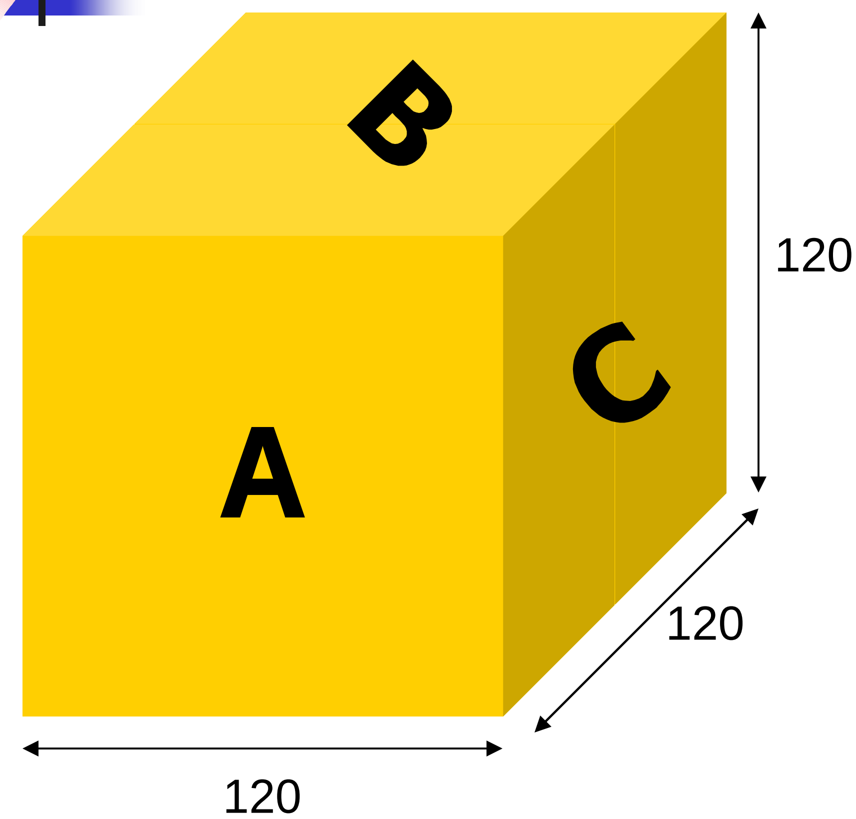
	Y[0]	Y[1]	Y[2]	Y[3]	Y[4]	Y[5]	Y[6]

(P0,S0) = LD(X[0],X[1])
(S1,P1) = LD(Y[1],Y[2])
(P2,S2) = LD(X[2],X[3])

(P3,S3) = P8*(P0,S0)+(P1,S1)
P3= P8*P2+P1

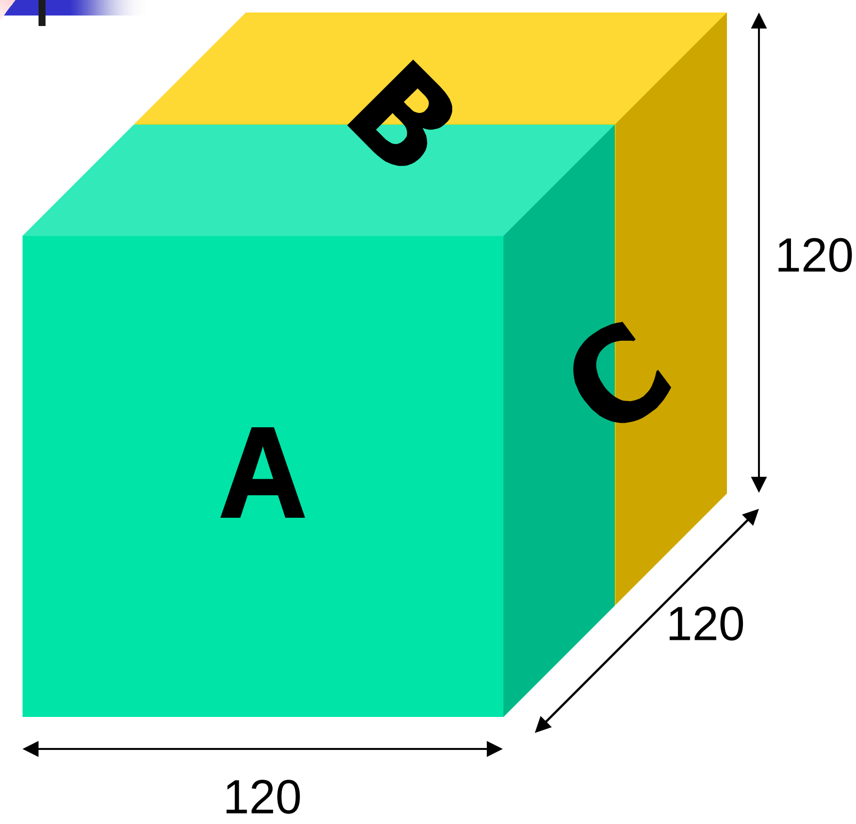
(Y[1],Y[2]) = ST(S3,P3)

Matrix Multiplication



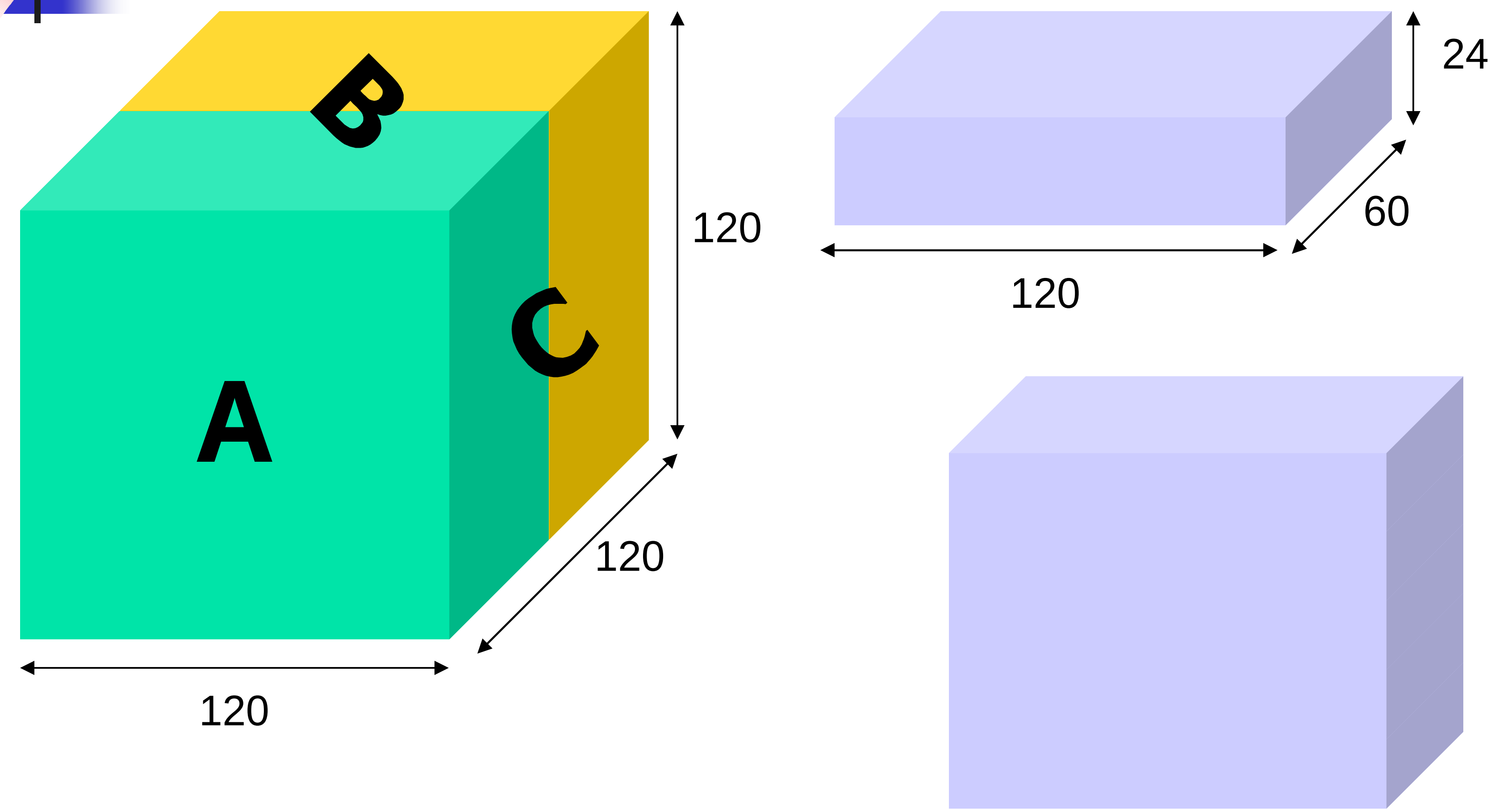
- Problem size chosen from L3 capacity constraints
- Three levels of tiling
  - For dual core
  - For L1 cache
  - For registers

Matrix Multiplication
Tiling for Dual Cores



- Lack of coherence in L1 dictates split of C
- B “streams” through L1: split it to control stream traffic
- Total data volume = 120×120×8×3 B = 345,600 B
  - Easily fits in L3 cache

Matrix Multiplication
Tiling for L1

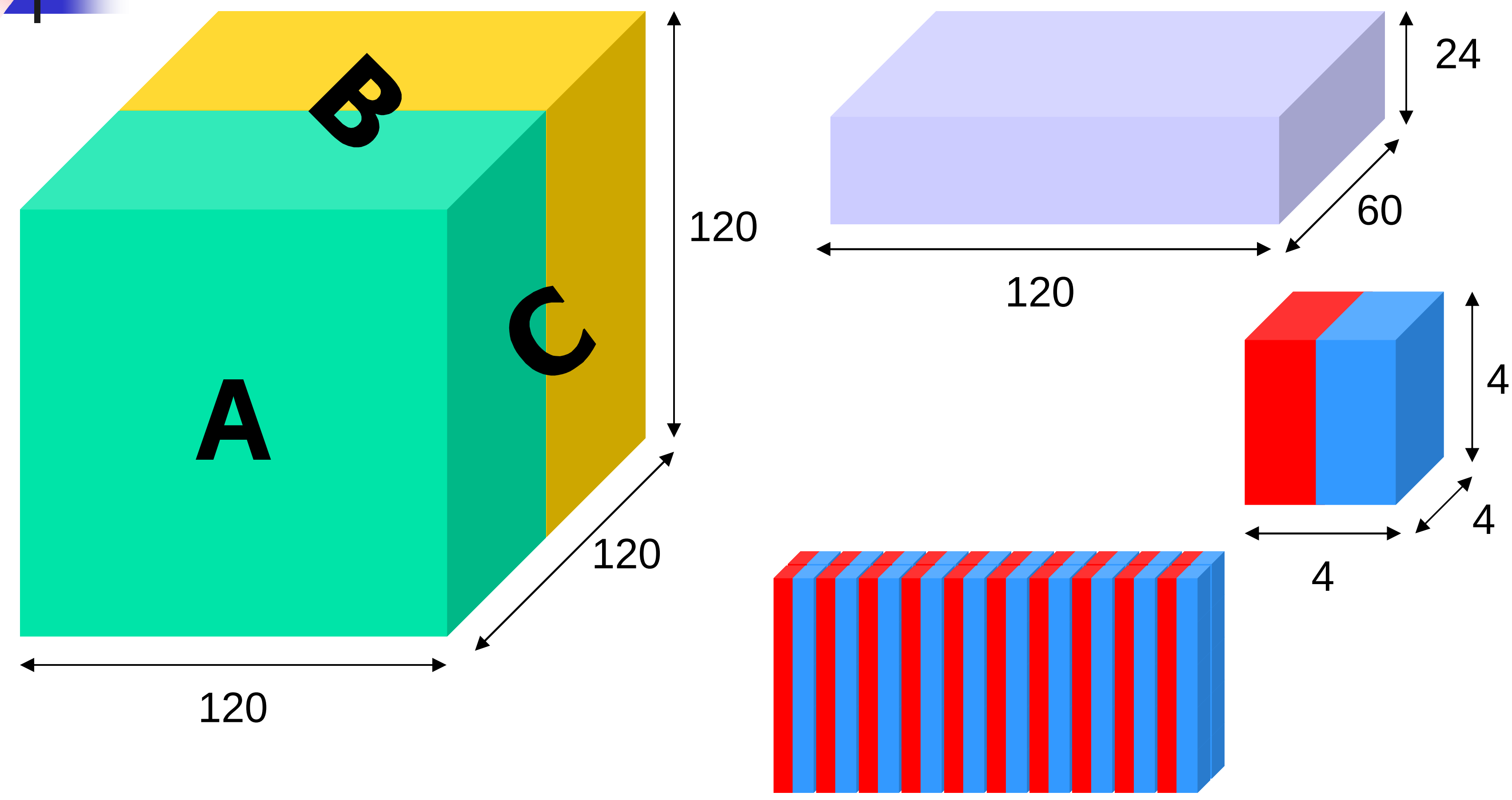


Matrix Multiplication
Tiling for L1 (Analysis)

- L1 holds 32KB = 4K elts = 1024 lines
  - Configured as 16 sets × 64 ways
- A occupies 24 × 120 elts = 2880 elts = 720 lines = 45 ways of L1 cache
- B streams through L1 in 4-col groups
  - 120 ×4 elts = 480 elts = 120 lines = 8 ways
- C is L3-hot, and loaded into registers
  - Some interference between A and C

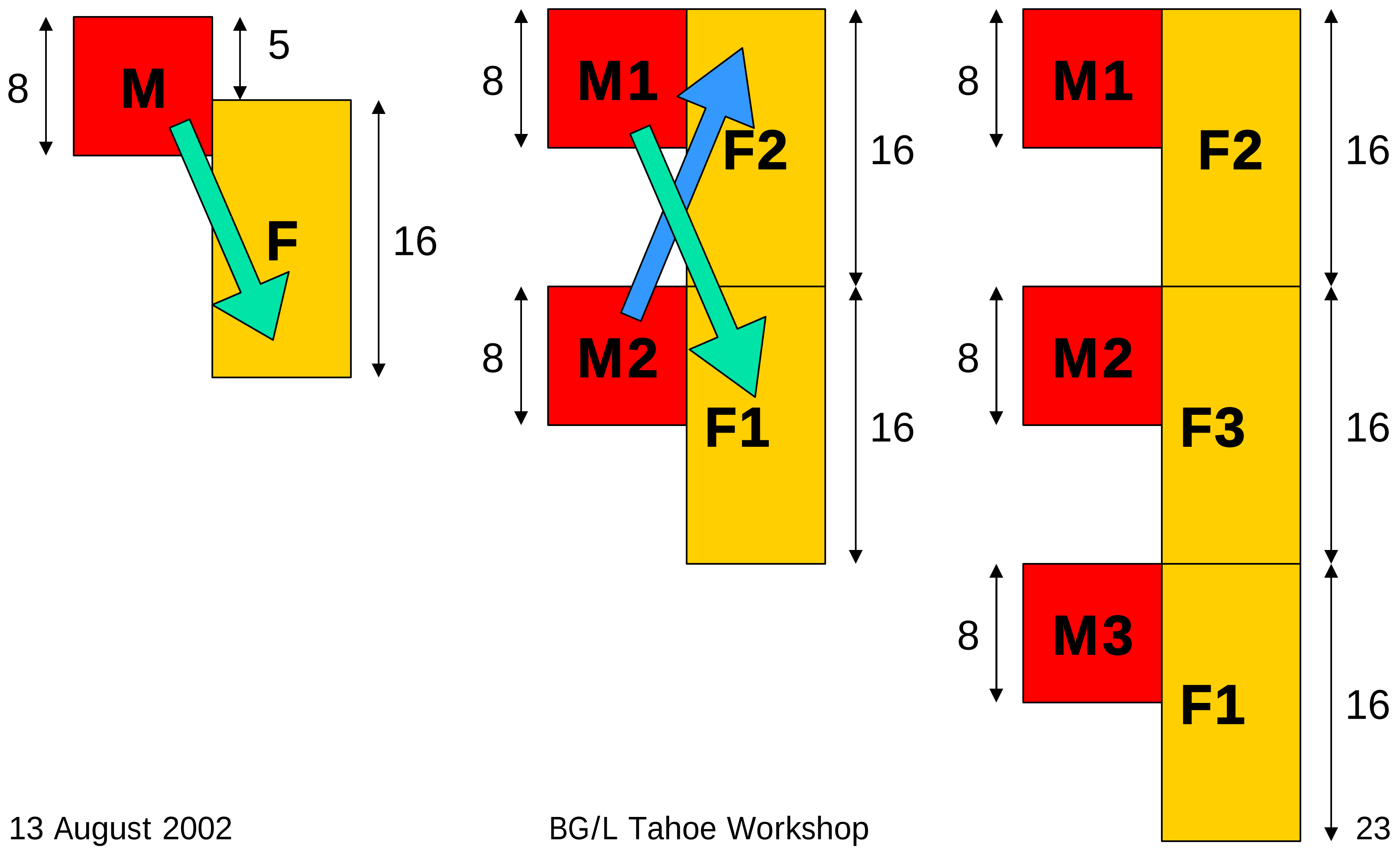
Matrix Multiplication

Tiling for Registers



Matrix Multiplication

Tiling for Registers (Dependences)



Matrix Multiplication
Tiling for Registers (Analysis)

- Usual kernel updates C(i:i+3,j:j+3) with outer product of A(i:i+3,k) and B(k,j:j+3)
- Change to A(i:i+3,k:k+1) and B(k:k+1,j:j+3) for double register file
  - 16 SIMD FMAs, eight QW loads, 16 register pairs
- Unroll by factor of two
  - 24 register pairs, 15 cycle load-to-use latency
- Could go to 3-way unroll if needed
  - 32 register pairs, 31 cycle load-to-use latency

Matrix Multiplication
Performance Results

- MTI simulation, Stage 7 model
- Single core (problem size: 24×16×58)
  - Optimal cycles = (24×16×58)/2 = 11136
  - A L1-hot, B and C DDR-hot
    - 15049 cycles, 74% of peak flops
  - A, B, C L1-hot
    - 12218 cycles, 91% of peak flops
- Dual core (problem size: 24×8×58 per core)
  - Optimal cycles = (24×8×58)/2 = 5568
  - A L1-hot, B and C DDR-hot
    - 7325 cycles, 76% of peak flops
  - A, B, C L1-hot
    - 5987 cycles, 93% of peak flops

Conclusions and Directions

- Preliminary idea of single-node performance programming strategies
  - Measurements for matrix multiplication
- Necessary future work
  - Systematic and more extensive measurements of memory access patterns
  - More complete analysis of other benchmarks
  - Performance models for linear algebra kernels
- Questions? Comments? Feedback?